Szitálás a prímtesztelésben és a faktorizációban

Vatai Emil

Doktori értekezés tézisei
Komputeralgebra Tanszék
Eötvös Loránd Tudományegyetem, Informatikai Kar

Témavezető:
Antal Járai D.Sc.

Informatika Doktori Iskola
Dr. Benczúr András
Numerikus és szimbolikus számítások
Dr. Antal Járai

Budapest, 2014

Tartalomjegyzék

1. Bevezetés és célkitűzések

A szitálás fontos szerepet játszik a számítógépes számelmélet algoritmusaiban [Bre89]. Szitákat használnak a prímkereső algoritmusok [FV13] és a faktorizációs algoritmusok is [Con97]. A szitálás előnye abban rejlik, hogy az (1) összeget ki tudjuk számolni anélkül, hogy minden i-re leellenőriznénk, hogy $p \mid i - q$ teljesül-e. Helyette megkeressük az első i_0 indexet, melyre ez teljesül és $f(p)$-vel növeljük az összes $i = i_0 + kp$ indexű tagot (ahol $k \in \mathbb{Z}$).

Eredetileg a faktorizációs algoritmusok szitájának gyorsítása volt a cél, viszont a végeredmény, egy általános „cache barát" szitáló algoritmus lett. Az algoritmus egy nagy szitatáblát szitál, hatékonyan adminisztrálva a (nagy) prímeket, és innen a „cache optimalizált lineáris szita" név (lásd [JV10, JV11]), vagy az angol elnevezés kezdőbetűiből COLS.

Más optimalizációk is figyelembe lettek véve, mint például, a párhuzamosítás. Az eredmény, az inverz szita (lásd [Vat13]) algoritmus, amely az Eratoszthenészi szitánál alkalmazható (a faktorizációs algoritmusoknál nem, ugyanis a szitáló művelet nem idempotens).

2. Alap definíciók

A fenti általános algoritmus tárgyalásához és helyesség bizonyításához a következő alapfogalmakat definiáljuk.

1. Definíció (Prím-offset vagy prím-eltolás párok)**.** *Adott P szitáló prímek halmazához tartozó prím-offset párok halmaza*

$$F = \{(p, q) \in P \times \mathbb{N} \colon 0 \leq q < p\}$$

úgy, hogy ha $(p, q) \in F$ és $(p, q') \in F$ akkor $q = q'$ és minden $p \in P$ prímhez létezik $q \in \mathbb{N}$, hogy $(p, q) \in F$.

2. Definíció (Szitatábla)**.** *Az M méretű S szitatábla (ahol $M \in \mathbb{N}$) egy M darab T halmaz elemeiből álló 0-tól $M - 1$-ig indexelt sorozat. Az i indexű elemet $S[i]$-vel jelöljük.*

3. Definíció (Szita művelet)**.** *A T halmazon értelmezett $\oplus\colon T \times T \to T$ műveletet szita műveletnek hívjuk, ha az asszociatív és kommutatív.*

A fenti definíciókkal a szitálást matematikailag definiálhatjuk a következő módon:

4. Definíció. *Az előző definíciók jelöléseivel, és adott $f\colon P \to T$ függvény esetén, az S szitatábla $(p, q) \in F$ prím-offset párral való szitálásának eredménye az S' szitatábla, ha*

$$S'[i] = \begin{cases} S[i] \oplus f(p) & \text{ha } \exists l \in \mathbb{N}, \text{ hogy } i = q + lp \\ S[i] & \text{különben} \end{cases}$$

Az S szitálása az $F' \subset F$ részhalmaz prímjeivel úgy kapható meg, ha S'-t szitáljuk $(p, q) \in F'$ párral, ahol S' tábla az $F' \setminus \{(p, q)\}$ részhalmazzal való szitálás eredménye. Ha $F' = \emptyset$, akkor a szitálás eredménye S, azaz változatlan marad.

Az előző definíció tükrözi a számítógépes program által megvalósított szitákat. Egy matematikailag jobban kezelhető, ekvivalens definíciót ad a következő lemma:

1. Lemma. *Ha az S táblát az F prím-offset halmazzal szitáljuk, akkor az eredmény S' melyre a következő teljesül:*

$$S'[i] = S[i] \oplus \sum_{\substack{(p,q)\in F \\ p|i-q}} f(p) \tag{1}$$

ahol a szumma a $\oplus$ szita műveletre vonatkozik.

3. Cache optimalizált lineáris szita

A szitálás nem illeszkedik jól a modern számítógépek memória hierarchiájához, ezért szükséges a szegmensenkénti szita használata, ahol egy szegmens belefér a cache memóriába, és így csökkenti az olyan adatokhoz való hozzáférést amelyek nincsenek a cache-ben.

5. Definíció (Szegmens)**.** *Legyen S egy M méretű szitatábla és $m \mid M$ ahol $m \in \mathbb{N}$. Az m hosszú, t-edik részsorozata S-nek, ahol a t index 0-tól $m-1$ értékig fut, az S szitatáblának t-edik szegmense és S_t-vel jelöljük, azaz $S_t[i] = S[mt+i]$ minden $0 \leq i < m$-re.*

Input: A már inicializált M méretű S szitatábla.
Input: Az m szegmens méret melyre igaz, hogy $m \mid M$ és $m \in \mathbb{N}$.
Input: Az F prím-offset párok halmaza.
Output: Az S tábla az összes F-beli prím-offset párokkal szitálva.

```
1 for t ← 0 ... m − 1 do
2     Az S_t előreolvasása a cache memóriába;
3     forall the (p, q) ∈ F do
4         while q < m do
5             S_t[q] ← S_t[q] ⊕ f(p) ;
6             q ← q + p;
7         A p-hez tartozó új q − m offset elmentése F-be, ami a
          következő szegmens szitálásakor felhasználandó;
```

Result: Az összes szegmens S_0-tól S_t-ig ki van szitálva és (1) igaz minden $0 \leq i < tm$-re.

Algoritmus 1: Szegmensenkénti szita

Az alap ötlet a szegmensenkénti szita hatékony megvalósításához a következő egyszerű lemmán alapszik:

2. Lemma. *Ha p egy szitáló prím, $km < p < (k+1)m$ (ahol $k \in \mathbb{N}$), és p az S_t szegmensbe szitál, akkor $S_{t'}$ a következő szegmens amelybe p szitálni fog, ahol $t' = t + k$ vagy $t' = t + k + 1$.*

Az előző lemmát, úgy is át tudjuk fogalmazni, hogy egy km és $(k+1)m$ közötti prím k vagy $k+1$ szegmenst ugrik át.

A COLS algoritmus úgynevezett kör és edény adatszerkezeteket használ a prím-offset párok hatékony kezelésére.

6. Definíció (Körök és edények)**.** *Legyen $K = \lfloor \max P/m \rfloor + 1$ a körök száma. Továbbá legyenek P_k és F_k halmazok a P-beli prímek illetve az F-beli prím-*

offset párok méret szerinti osztályozása, ahol $0 \leq k < K$ *és*

$$P_k = \{p \in P \colon km \leq p < (k+1)m\}$$
$$F_k = \{(p,q) \in F \colon km \leq p < (k+1)m\}$$

Legyen $0 \leq d \leq k$*. A kezdő* $(t = 0)$ *állapotban lévő,* d *indexű,* k*-ad rendű edényt,* $B^0_{k,d}$*-val jelöljük, és olyan prím-offset párokat tartalmaz, hogy*

$$B^0_{k,d} = \{(p, q - md) \colon (p,q) \in F_k \wedge dm \leq q < (d+1)m\}$$

A $t+1$ *állapotban lévő* d *indexű* $0 < k$*-ad rendű edény (ahol* $0 \leq t < M/m$*)*

$$B^{t+1}_{k,d} = \begin{cases} \{(p,q) \colon (p,q) \in \overline{B}^t_{k,b} \wedge 0 \leq q\} & \text{ha } d \equiv t \pmod{k+1} \\ B^t_{k,d} \cup \{(p, q+m) \colon (p,q) \in \overline{B}^t_{k,b} \wedge q < 0\} & \text{ha } d \equiv t-1 \not\equiv t \pmod{k+1} \\ B^t_{k,d} & \text{különben} \end{cases}$$

ahol

$$\overline{B}^t_{k,b} = \{(p, q + \delta p - (k+1)m) \colon (p,q) \in B^t_{k,b}\}$$
$$\delta = \min\{l \in \mathbb{N} \colon q + lp \geq m\}$$
$$b = t \bmod (k+1)$$

és b *az aktuális edény indexét jelöli.*

A $C^t_k = \{B^t_{k,d} \colon 0 \leq d \leq k\}$ *edények halmaza alkotja a* t *állapotban lévő* k*-ad rendű kört.*

1. Tétel. *Minden* $0 \leq t \leq M/m$*-re és* $0 \leq k < K$*-ra a* P_k *és* C^t_k *halmazok egyenlőek, olyan értelemben, hogy minden* $p \in P_k$*-hez van pontosan egy olyan* (p, q') *prím-offset par, hogy pontosan egy* $B^t_{k,d}$ *edényben szerepel amely a* C^t_k *körhöz tartozik (valamilyen* $0 \leq q' < m$ *offsettel és* $0 \leq d \leq k$ *indexszel).*

A következő invariánsok teljesülnek a COLS algoritmus futása közben:

2. Tétel. *Minden* $0 \leq k < K$ *indexre és* $0 \leq t \leq M/m$ *állapotra a* $B^t_{k,d}$ *edény pontosan azokat a* (p, q') *párokat tartalmazza, melyekre a következő állítások*

teljesülnek:

$$
\begin{aligned}
& km \le p < (k+1)m \\
& 0 \le q' < \min\{p, m\} \\
& q \equiv (t+d-b+k+1)m + q' \pmod{p} && \text{ha } 0 \le d < b \\
& q \equiv (t+d-b)m + q' \pmod{p} && \text{ha } b \le d \le k
\end{aligned}
$$

ahol $b = t \bmod (k+1)$ *az aktuális edény indexe és* $(p, q) \in F$.

1. Következmény. *Az aktuális edény prímjei mindig az aktuális* S_t *szegmenst szitálják.*

2. Következmény. *A* $B^t_{k,d}$ *edény pontosan azokat a prímeket tartalmazza, melyek az* $S_{t+d'}$ *szegmenst szitálják, ahol* $d' = (d+k+1-b) \bmod (k+1)$.

A COLS a 2. algoritmusban van összefoglalva.

Input: Inicializált S szitatábla, a 6 definíciónak megfelelően körökbe és edényekbe rendezett prím-offset párok.
Output: Az összes prím-offset párral szitált S tábla.

```
1 for t ← 0 ... M/n − 1 do
      /* A C_0 beli prímekkel másképp szitálunk.                    */
2     Hajtsuk végre az 1 algoritmus belső ciklusát C_0-ra;
3     for k ← 1 ... K − 1 do
4         for (p, q) ∈ B^t_{k,b} do
5             S_t[q] ← S_t[q] ⊕ f(p);
6             Tegyük az új q' offsettet a megfelelő edénybe.;
              /* b a C_k kör aktuális edénye.                       */
7             b ← (b + 1) mod (k + 1); /* A C_k kör forgatása.     */
```

Algoritmus 2: A cache optimalizált lineáris szita áttekintése

A COLS legbelső ciklusának optimalizálása assembly nyelvben van megadva a következő programkódban.

```
1 bts    [RDX],R13d          ; sieve at offset q
2 lea    R13d,[R12d+R13d]    ; calculate q+p
3 lea    R10d,[R13d+R8d]     ; store LO offset (temp.)
```

```
4  sub   R13d,R9d            ; store HI offset
5  mov   RCX,RSI             ; store HI address
6  cmovc RCX,RDI         ; rewrite to LO address (cond.)
7  cmovc R13d,R10d       ; rewrite to LO offset (cond.)
8  setc  BL                  ; if LO : RBX=1 else 0
9  add   RBX,RBX             ; if LO : RBX=2 else 0
10 mov   [RCX],R12d          ; save prime (to LO or HI)
11 mov   [RCX+4],R13d        ; save offset (to LO or HI)
12 lea   RSI,[RSI+RBX*4-8]   ; HI : RSI=HIaddr-=8 or
13 lea   RDI,[RDI+RBX*4]     ; LO : RDI=LOaddr+=8
14 mov   R14d,[RCX+RBX*8-8] ; load next prime
15 mov   R15d,[RCX+RBX*8-4] ; load next offset
```

Listing 1. A `sieve_b()` implementációja x86 assembly nyelven

4. Az inverz szita

Az áram disszipáció miatt, a processzorok fejlesztése zsákutcába jutott az órajel növelés szempontjából. Az utóbbi pár évben, a legtöbb gyártó tartja a körülbelül 3GHz órajellel rendelkező processzorok gyártását, és csak a magok számát növelik. A sebesség megtartásához a legtöbb programot át kell írni, figyelembe véve a párhuzamosítást is. Az igazán nagy feladatok megoldásához a munkát több számítógépek között is fel kell osztani.

Az alap megközelítés abból áll, hogy különböző prímeket küldünk különböző szitáló nódusokra és a végén összefésüljük az így kapott (részben) kiszitált táblákat; viszont, ez komoly kommunikációs terhelést okoz, ami negálja az osztott végrehajtás előnyeit. Ezt a problémát az inverz szita úgy oldja meg, hogy egy nagyon egyszerű veszteséges tömörítést használva drasztikusan csökkenti a folyamatok közötti kommunikációt, amellett, hogy szinte semmi többlet munkába nem kerül a tömörített tábla szitálása.

7. Definíció. *Legyen $m \mid M$ és S az M méretű szitatábla, ahol nullák reprezentálják a kiszitált elemeket és egyesek a potenciális kandidátusokat. Ebben*

az esetben a tömörített szitatábla

$$\hat{S}[k] = \begin{cases} 0 & \textit{ha } S[km+i] = 0 \textit{ minden } 0 \leq i < m\textit{-re} \\ 1 & \textit{különben} \end{cases} \tag{2}$$

A tömörítésre úgy gondolhatunk, mint ha az eredeti táblát felosztanánk m hosszú részintervallumokra és minden részintervallum tartalmát „szétmaszatolnánk". Így a tömörített tábla elemei maszatok, ahol az 1-es maszat olyan részintervallumnak felel meg az eredeti táblában, amely legalább egy 1-es bitet tartalmaz; és a 0-ás maszat, olyan részintervallumnak felel meg, amelyben minden elem ki van szitálva, azaz csak 0-kat tartalmaz.

Miután a kis prímekkel szitálunk, melyek a kandidátusok zömét kiiktatják, a szitatábla hosszú 0-ból álló intervallumokat fog tartalmazni. Ennek eredményeként a tömörített táblában is sok 0 maszat fog szerepelni.

Input: Tömörített $\hat{S}$ szitatábla M/m darab maszattal, egy F_j prím-offset párok halmaza.
Output: Elmentett offsetek, amelyeket majd visszaküld összefésülésre.

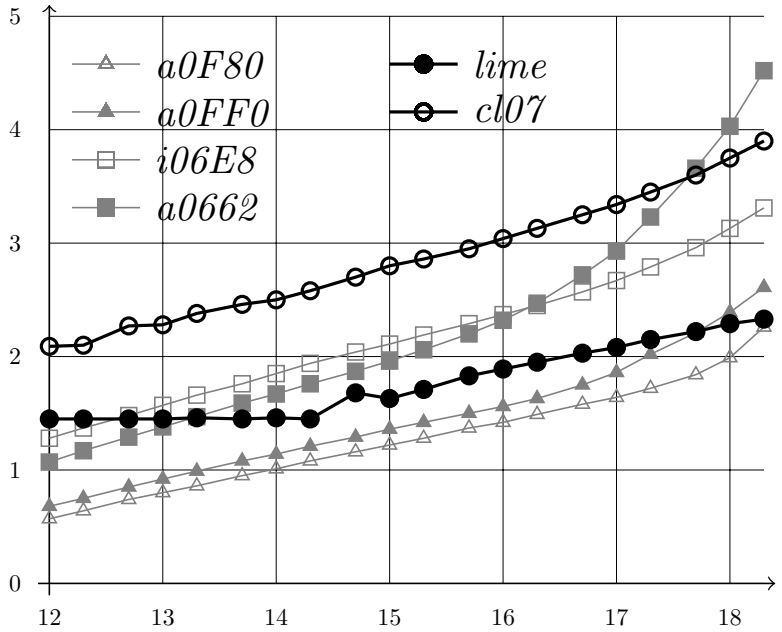
```
1 foreach (p, q) ∈ F_j do
2     while q < M do
3         q' ← ⌊q/m⌋ ;          /* ami csak q' ← q' ≪ c ha m = 2^c */
4         if Ŝ[q'] = 1 then Save(q);
5         q ← q + p;
```

Algoritmus 3: Az inverz szita

Az inverz szitáló számítógépek eredményként azokat az offseteket küldik vissza (egészek tömbjeként tárolva), melyek lehet, hogy szitálni fognak az eredeti táblában. Az inverz szita előnye az, hogy eliminálja azokat az offseteket, melyek biztos nem szitálnak, míg a kommunikációt minimumon tartja.

5. Eredmények

1. Eredmény. Egy általános keretrendszer szitáló algoritmusokra és a COLS algoritmus helyességének bizonyítása.



1. ábra. Algoritmusok futási idők összehasonlítása

2. Eredmény. Egy számítógépes program, amely az Eratoszthenészi szitát hajtja végre 10^{17} nagyságrendű számok $2^{30} \approx 10^9$ hosszú intervallumán, melyet egy optimalizált programmal [eS11] hasonlítottunk össze. Az eredmény a 1. ábrán van ábrázolva (a „lime" és „cl07" gépeken fut a COLS algoritmus).

Megjegyzés. A hardverbeli különbségek miatt, az eredményeket nehéz összehasonlítani, viszont figyelemre méltó javulás tapasztalható a lassú memóriával rendelkező gépeknél. Ez nagyon fontos az egyre növekvő processzor és a memória sebesség közötti különbség miatt.

3. Eredmény. Az inverz szita, azaz egy javaslat a szitálás osztott megvalósítására, amely saját memóriával és korlátozott sávszélességgel rendelkező gépeken fut.

A [JV11] cikkben leírt COLS algoritmust implementálták a [Sut14] szitában és idézték a [CSB14] cikkben.

Hivatkozások

[Bre89] David M. Bressoud. *Factorization and Primality Testing*. Undergraduate Texts in Mathematics. Springer Verlag, 1989. 1

[Con97] Scott P. Contini. Factoring integers with the self initializing quadratic sieve. Master's thesis, University of Wisconsin - Milwaukee, 1997. 1

[CSB14] CarlosM. Costa, AltinoM. Sampaio, and JorgeG. Barbosa. Distributed prime sieve in heterogeneous computer clusters. In Beniamino Murgante, Sanjay Misra, AnaMariaA.C. Rocha, Carmelo Torre, JorgeGustavo Rocha, MariaIrene Falcão, David Taniar, BernadyO. Apduhan, and Osvaldo Gervasi, editors, *Computational Science and Its Applications – ICCSA 2014*, volume 8582 of *Lecture Notes in Computer Science*, pages 592–606. Springer International Publishing, 2014. 5

[eS11] Tomás Oliveira e Silva. Goldbach conjecture verification. `http://www.ieeta.pt/~tos/goldbach.html`, 2011. 2

[FV13] Gábor Farkas and Emil Vatai. Sieving for large cunningham chains of length 3 of the first kind. *Annales Univ. Sci. Budapest, Sect. Comp.*, 40:215–222, 2013. 1

[JV10] Antal Járai and Emil Vatai. Cache optimized sieve. In Horia F. Pop and Antal Bege, editors, *8th Joint Conference on Mathematics and Computer Science*, pages 249–256, Komárno, Slovakia, 2010. 1

[JV11] Antal Járai and Emil Vatai. Cache optimized linear sieve. *Acta Univ. Sapientiae, Inform.*, 3:205–223, 2011. 1, 5

[Sut14] Jared Suttles. primesieve - optimized sieve of eratosthenes implementation. `https://github.com/jaredks/pyprimesieve/tree/master/primesieve`, 2014. 5

[Vat13] Emil Vatai. Inverse sieve. *Annales Univ. Sci. Budapest, Sect. Comp.*, 41:355–360, 2013. 1